

SIVO/ASTG Process Standard

Guides and Procedures for Operations

AstgProcessStandard.doc

Version 1.6

September 12, 2011

Contents

1 INTRODUCTION	1
1.1 DOCUMENT OVERVIEW	1
1.2 INTENT	1
1.3 PROJECTS OVERVIEW	1
1.3.1 GMI.....	2
1.3.2 MAP Workflow Tool	2
1.3.3 Other Projects	2
1.3.4 Level II User Support.....	4
2 PROJECT MANAGEMENT GUIDE	5
2.1 AGILE SOFTWARE DEVELOPMENT	5
2.2 SOFTWARE LIFECYCLE	5
2.3 PROJECT PLANNING	6
2.3.1 <i>Iterative and Incremental Development</i>	6
Lifecycle.....	6
Iteration Length	6
Timeboxing	6
2.3.2 <i>Adaptive Planning</i>	6
Level of Planning Detail	6
Activity Focus	7
Client-Driven.....	7
Risk-Driven	7
Evolutionary Delivery	7
2.3.3 <i>Estimation</i>	7
2.3.4 <i>Tracking</i>	8
Metrics.....	8
Backlogs	8
2.4 CUSTOMER REQUIREMENTS	8
2.4.1 <i>Evolutionary Requirements</i>	8
Lifecycle.....	8
Focus	9
2.4.2 <i>Capture Methods</i>	9
Other Work Products.....	9
3 SOFTWARE DEVELOPMENT GUIDE	10
3.1 SOFTWARE DEVELOPMENT PROCEDURE	10
3.1.1 <i>Understand the Requirements</i>	10
3.1.2 <i>Track the Request</i>	10
3.1.3 <i>Design the Software</i>	11
3.1.4 <i>Check Out the Code</i>	11
3.1.5 <i>Code the Changes</i>	11
3.1.6 <i>Test and Validate the Code</i>	11
3.1.7 <i>Check In the Code</i>	14

3.1.8 <i>Release the Code</i>	14
3.1.9 <i>Doing Transition and Maintenance</i>	15
3.2 RECOMMENDED DEVELOPMENT PRACTICES	15
3.2.1 <i>UML</i>	15
3.2.2 <i>Design Patterns</i>	15
3.2.3 <i>Inspections</i>	15
3.2.4 <i>Code Reviews</i>	16
3.2.5 <i>Pair Programming</i>	16
3.2.6 <i>Automated Documentation</i>	16
3.2.7 <i>Test-Driven Development</i>	16
3.2.8 <i>CVS Tips and Practices</i>	17
4 PRODUCTION RUN GUIDE	19
4.1 PRODUCTION RUN PROCEDURE	19
4.1.1 <i>Evaluate the Request</i>	19
4.1.2 <i>Get the Requirements</i>	19
4.1.3 <i>Prepare for the Run</i>	19
4.1.4 <i>Examine and document the outputs</i>	20
5 LEVEL II USER SUPPORT GUIDE	22
5.1 OBJECTIVE	22
5.2 TECHNICAL APPROACH	22
5.2.1 <i>Track the Request</i>	22
5.2.2 <i>Diagnose the Problem</i>	22
5.2.3 <i>Determine the Resolution</i>	23
5.2.4 <i>Closing a Ticket</i>	23
6 WEB SITE DEVELOPMENT GUIDE	24
6.1 OBJECTIVE	24
6.2 DEVELOPMENT PROCEDURE	24
6.2.1 <i>Preparations</i>	24
6.2.2 <i>Creating Pages</i>	24
6.2.3 <i>Web Site Approval Process</i>	25
6.2.4 <i>Publishing</i>	25
6.3 GMI WEB SITE SPECIAL OPERATIONS	26
6.3.1 <i>Schedule</i>	26
6.3.2 <i>Procedures</i>	26
6.4 PRINT-PROTECTING GMI PDFs	26
7 APPENDIX	28
7.1 REFERENCES	28
7.1.1 <i>GMI Project</i>	28
7.1.2 <i>CVS</i>	28
7.1.3 <i>Website Development</i>	28

7.1.4 <i>Software Development</i>	29
7.2 COMMON TOOLS	29
7.3 NASA STI APPROVAL FOR JOURNAL/CONFERENCE PAPERS	30

1 Introduction

1.1 Document Overview

This document has been broken into several sections that cover different aspects of software development processes and general operations that our team performs for NASA's Software integration and Visualization Office (SIVO).

1. The **Project Management Guide** is intended for those who must plan how to direct the work of the project. Sometimes that will fall on the developers of the system and other times it may be a separate role. The principles described here are also important for any software developer.
2. The **Software Development Guide** is intended for those who will design and implement the system. It contains procedures we are currently using and recommended practices for future development.
3. The **Production Run Guide** is intended for those who will perform production runs of NASA models. It contains procedures we are currently using.
4. The **Level II User Support Guide** describes the basic procedures used in providing tech support to SIVO customers. It contains procedures we are currently using.
5. The **Web Site Development Guide** examines the processes required to create, and get approval for a new set of pages on the NASA GSFC web server. It also contains special procedures needed when updating "access-restricted" material.

1.2 Intent

This process standard document provides the principles, guidance and procedures to be used during a SIVO software project. It covers the range of software lifecycle activities from requirements analysis to release procedures and support operations. The information contained in this standard applies to both project management and technical activities.

The use of process standards adopted by the team provides a number of benefits, including:

1. Provide a common operational background for team members
2. Incorporate shared team process experience
3. Act as a guide to new team members
4. Make development more predictable and repeatable
5. Save time by avoiding mistakes
6. Maintain high quality and lower risk

1.3 Projects Overview

The following is a brief description of the projects or tasks currently supported or recently completed by the ASTG team:

1.3.1 GMI

The **Global Modeling Initiative** (GMI) project includes developing and maintaining a state-of-the-art modular 3-D chemistry and transport model that can be used to assess the impact of various natural and anthropogenic perturbations on atmospheric composition and chemistry. The GMI model also serves as a testbed for model improvements. GMI has developed a modular chemical-transport model with the ability to incorporate different components and inputs, such as meteorological fields, chemical and microphysical mechanisms, numerical methods, source gas emissions, and other modules representing different approaches of current models in the scientific community, as well as carry out multiyear assessment simulations (<http://gmi.gsfc.nasa.gov>).

ASTG is involved in implementing requested science upgrades, refactoring code to improve performance, doing production runs for differing sets of input conditions and chemical mechanisms, and supporting users of the system.

ASTG also assists scientists in the implementation of new chemical mechanisms in the code (for instance the combined stratospheric/tropospheric mechanism). In addition, they provide support in the pre-processing step for generating Fortran routines for internal computations in the chemical mechanism.

1.3.2 MAP Workflow Tool

ASTG Developed the Workflow Tool to assist NASA scientists with managing complex model experiments. This advanced system allows users to configure, monitor, and manage experiments via an easy-to-use GUI, replacing the error-prone process of configuring scripts and executing a series of system commands. The workflow system takes care of scheduling and executing the model run tasks in the proper order. Such tasks include source check-out, building the executable, execution, post-processing, visualization, etc.

The Workflow Tool GUI application is known as the NASA Experiment Designer (NED). NED's generic interface was designed to work with many models and computing systems. So far, NED has been customized to work with GMI, GEOS-5, and the Sensor Web Simulator (see Other Projects below). Developers can adapt the Workflow Tool to work with their own model by writing a series of Python scripts that define the process for NED to control. Further information on using the tool can be found at: <http://modelingguru.nasa.gov/clearspace/community/mapmewkflow>. Latest efforts involve developing a Java web-application version that requires no installation, and can be run through standard web browsers.

1.3.3 Other Projects

1. **GISS ModelE** is the primary climate model developed and maintained at the Goddard Institute for Space Studies (GISS). ASTG is using the Message Passing Interface (MPI) and the Earth System Modeling Framework (ESMF) both to parallelize modelE for multi-processor high-performance architectures as well as to enable the use of the Lin-Rood finite-volume (FV) dynamical core. The FV dynamical core has been wrapped as an ESMF component by the Global Modeling and Assimilation Office (GMAO) and thereby made usable by other modeling efforts such as that at GISS. Incorporation of the FV core within modelE involves substantial re-engineering of the modelE design.

2. **Gigaparticle Trajectories** – This particle trajectory program was recently converted from run-time-interpreted IDL code to compiled C++ code. The new version was written in C++ to allow the program to be implemented as an extensible application framework and to make use of the faster execution of a compiled language. It is also parallelized using the Message Passing Interface (MPI) library to allow the simulation of 1 billion trajectories on a suitable NASA computing platform.
3. **Chemical Transport Model (CTM) Framework** – The main objective of this work is to make CTMs more flexible, more accessible, modular, and ESMF compliant. A modular framework can offer the ability to more effectively incorporate different components and inputs into CTMs. Among these components and inputs are: various meteorological fields, chemical and microphysical mechanisms, numerical methods, source gas emissions, and other modules representing the different approaches in current CTM models. In our approach we (1) Identify all the major components in the following models: GMI, Goddard Earth Observing System (GEOS)-Chem, Goddard 3D CTM, and Global Ozone Chemistry Aerosol Radiation Transport (GOCART); (2) Isolate each component by making them self-contained (process of componentization); (3) Design interfaces to invoke each component with the proper import and export information. This process produces a flexible framework that allows interchangeable CTM components while preserving each model's integrity.
4. **Earth System Modeling Framework (ESMF)** – ASTG is providing support for integration of weather and climate models, including the Goddard Cumulus Ensemble (GCE) model, NASA GMAO data assimilation system for precipitation, and GMI model. In addition, they're supporting ESMF framework development in extending the features, fixing bugs, and proposing new features for SIVO customers.
5. **Message Passing Interface (MPI) I/O Optimization** – ASTG is optimizing I/O for the GCE model, GISS ModelE, and GEOS-5 Atmospheric General Circulation Model. They are utilizing concurrency and better access time to memory vs. disc to develop a MPI-based I/O library.
6. **Sensor Web Simulator** – This project will: (i) design a sensor web architecture that couples Earth observing systems with scientific models and data assimilation systems; and (ii) build an end-to-end sensor web simulator (SWS) based upon the proposed architecture that would objectively assess the scientific value of a fully functional model-driven meteorological sensor web.
7. **Climate@Home** – ASTG is creating a software framework to support a virtual supercomputing network that spreads the data-processing chores across thousands of computers. Similar to SETI@Home, this technology eliminates the need to buy additional supercomputers, which consume enormous amounts of energy, by allowing volunteers to use their excess computing power to run climate model experiments. Climate@Home plans to test the accuracy of ModelE developed by the Goddard Institute of Space Studies (GISS) in New York, and will serve as a trailblazer to explore the accuracy of other models as well.
8. **NU-WRF** – The NASA Unified Weather Research and Forecasting (NU-WRF) Model is an effort to unify WRF with NASA's existing weather models and assimilation systems, such as GEOS-5 and LIS. Several parameterizations of physical processes developed by

NASA scientists have been implemented into NU-WRF to better represent/simulate cloud-aerosol-precipitation-land surface processes. By developing a NASA unified WRF, the modeling community expects to: a) facilitate better use of WRF for scientific research, b) reduce redundancy in major WRF development, c) prolong the serviceable life span of WRF, and d) allow better use of NASA high-resolution satellite data for research (short term climate and weather).

1.3.4 Level II User Support

The ASTG provides expert troubleshooting support for users of the NASA Center for Computational Sciences (NCCS) and NASA Advanced Supercomputing (NAS) High-End Computing (HEC) supercomputers when the Level I Help Desk cannot resolve the issue. They assist users in the proper use of programming languages, compilers, debuggers, queuing systems, parallel programming packages, and model configuration/setup issues.

To enhance our support to NASA's modeling community, ASTG implemented and continues to provide moderation for an online knowledge base (KB) known as Modeling Guru, which is available at <https://modelingguru.nasa.gov>. This platform simplifies knowledge sharing and collaboration among NASA scientists, while also benefiting from the experience of external researchers, at government, research and educational institutions. By supplying common answers to problems in modeling and usage of NASA HEC assets, we provide cost-effective application support and make users more self-sufficient.

2 Project Management Guide

Managing a project can be easy or difficult, depending on the complexity and people involved. The following practices are highly recommended because they are being employed with great success by modern software projects. As with all software projects, the range of “formality” will vary, depending on the customer’s needs and the difficulty of the project.

2.1 Agile Software Development

Agile software development methods are well suited to the evolutionary nature and uncertainty inherent in software development. Traditional methods like the *waterfall method* have demonstrated poor performance due to a failure to address change and feedback throughout the project. Although Agile methods prefer “just enough” formality, this can be high in some circumstances, such as life critical systems. Agile is **not** ad-hoc or unstructured, but is “delivery and quality-oriented.”

Agile methods emphasize close collaboration between the programmer team and project sponsor, face-to-face communication (as more efficient than written documentation), frequent delivery of new deployable working code, and tight, self-organizing teams. The Agile Alliance has written a manifesto which sums up Agile development values:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

While the items on the right certainly have value, they should not override those on the left.

2.2 Software Lifecycle

Agile software development uses an iterative or “staged” delivery model. The main difference in doing iterative development is that most of the traditional phases of software development are performed in nearly every iteration! In other words, each iteration in the lifecycle may contain some amount of planning, requirements analysis, coding, and testing. Obviously, the earliest iterations will focus on discovering requirements, while later ones will focus on implementing features.

The phases in our software development model are:

1. **Software Concept** – Generally a single and shorter iteration to present the high-level objectives, collect and itemize the high-level requirements of the system and identify the basic functions and risks of the system.
2. **Elaboration** – This phase develops a high-level software architecture that outlines the functions, relationships, and interfaces for major components. It may last about three it-

erations and concentrate on the development of the basic requirements from the Software Concept phase. The requirements will also continue to be elaborated and by the end of the phase, should account for 80% of the major requirements.

3. **Construction** – After the architecture and requirements are more stable, construction begins in earnest. Construction consists of release cycles broken into iterations. Each iteration may include some amount of planning, requirements and requirements analysis, along with standard coding and testing. Each iteration results in a tested and integrated system. Planning begins before the next iteration starts. The development of release materials should also be considered.
4. **Transition** – Once the software is mature and the major development effort winds down, maintenance will begin based on user's needs.

The following sections in this document will expound on these steps and provide further guidance for each of the activities within an iteration and provide guidance for successful development using this lifecycle. It's important to remember that a main tenant of Agile projects is that requirements, plans, and processes should all be fluid, and should be adjusted to meet evolving goals through close collaboration with all parties involved.

2.3 Project Planning

2.3.1 Iterative and Incremental Development

Lifecycle

The software development lifecycle shall occur in several iterations with fully tested and integrated software at the end of the iteration. Every iteration should have some amount of planning, requirements, testing, coding and integration. Planning for the next iteration should occur before the iteration has started in earnest.

Iteration Length

The recommended iteration length shall be one to six weeks during which a partial system will be grown incrementally. This system is not a prototype or proof-of-concept exploration, but an actual subset of the final system.

Timeboxing

Iterations and release deliveries shall be “timeboxed” – or fixed to a specific date. The date of the iteration or delivery will not change, although the scope of the features to be delivered can be reduced to meet the delivery date.

2.3.2 Adaptive Planning

Level of Planning Detail

The first few iterations should merely focus on high level architecture, leaving future iterations with imprecise details. Detailed planning shall only occur on the next few iterations. Heavily detailed schedules created up-front rarely consider all the complexities and changes involved in a project, and can thus be highly risky and wasteful.

Several iterations shall be completed before a complete detailed plan is created if such is required. This allows feedback on how future plans might unfold.

Activity Focus

Early iterations shall focus heavily on exploring the requirements and creating a system architecture. Later iterations shall focus more heavily on testing and code development.

Client-Driven

The customer (or a representative) shall select the requirements or features for development in the next iteration or release. This provides them with the most business value early. Customers use information from previous iterations to plan what to implement for the next iterations. Customers may not change the selections for the current iteration without removing something else.

Risk-Driven

The developers shall select the most difficult and riskiest elements for the initial iterations. This resolves potential problems early. Developers may reduce the scope of the development based on progress. Early iterations should focus on establishing and proving the architecture.

Evolutionary Delivery

All teams shall integrate software periodically into a release. Recommended release frequencies are every three months to a year.

At the end of a release, a product demo will be delivered to the client for feedback. The iteration will be briefly reviewed for improvements and lessons learned.

2.3.3 Estimation

Inaccurate estimates are worse than useless because they produce wrong expectations. Most software development programs overshoot their estimates. Unfortunately, it is difficult to make accurate estimations at the beginning of a project. However, near-term tasks or projects that have begun in earnest (10-20% complete) can produce better results.

The following estimation method is recommended for most projects when estimates are required.

1. Produce detailed estimates for only the next few iterations, or wait until the project has completed at least three iterations.
2. When creating an estimate for an iteration, make a Task List that identifies the items that need to be completed to fulfill the selected requirements.
3. Estimate your level of effort for the project this week. Take into consideration average workload for other projects and other items related to accomplishing the tasks.
4. Spend time considering the level of effort needed for each task. Each task should be no longer than a 0.5 - 1.5 days. If it is longer, break it into smaller sub-tasks. The level of effort should have an (1) optimal, (2) pessimistic and (3) most-likely level of effort specified.

Use the following formula to produce a probable level of effort:

$$\frac{\text{optimistic} + \text{pessimistic} + (\text{most_likely} \times 4)}{6}$$

5. Consider using other experienced developers to create independent estimates. Then discuss why they are different and combine them.

2.3.4 Tracking

Tracking involves accounting how a project is following its plan and meeting its schedule, costs and quality targets.

Metrics

One way to track technical progress is the use of metrics. There are a number of variables that can be used to track progress. Some suggested metrics include number of modules, classes, or tests (total vs passed), if using test-driven development. Metrics evaluation should ideally be automated and updated regularly.

Backlogs

Backlogs are lists of features to be added to a software product. They help identify features (or use cases, requirements, bugs, etc.) that are to be implemented next. The backlogs are prioritized according to business value (client-driven) and risk (developer-driven) and updated regularly.

There are three levels of backlogs:

1. **Product Backlog** – lists features to be added to the product
2. **Release Backlog** – lists features to be implemented for the next release
3. **Sprint Backlog** – lists features to be implemented in the current (or next) iteration

In practice, ASTG employs the sprint backlog and product backlog on most projects and tracks these using Google spreadsheets, which are shared with all members of the project. The sprint backlog retains the list of completed items and shows the iteration cycle when it was completed (see example in Fig. 2.1 below)

Story Number	(Active; Complete; Waiting; Inactive)	Story	Magnitude	Tasks/Notes	Iteration Number (when completed)
MG.27	I	server	3	authentication & permissions work as expected.	
MG.24e	A	Import MultipleApprovers patch into dev SBS 4	8	Modify DbDocument.java source code (MultipleApprover patch) from 2.5 and modify as necessary to work with 4.X	
MG.26	A	Install, test, and document Openfire & Fastpath live chat support on SBS 4.0	100	Install Openfire/Fastpath on the SBS 4 dev server running CentOS5.4 and fully document the procedure.	
Completed					
MG.24c	C	Import Newbie Interceptor into dev SBS 4	3	Import current Newbie Interceptor from 2.5 and modify as necessary to work with 4.X	71
MG.24d	C	Import NASA theme templates into dev SBS 4	5	Import template files from 2.5 and tweak them to work with 4.X	70
MG.25	C	Complete NASA STI Document Availability Authorization (DAA) for ModGu	8	Work with Lara to complete NASA Form 1676 and GSFC Forms 25-49 & 25-53 to get blanket authorization for Scientific & Technical Information (STI) posted on ModGu.	70

Fig 2.1 ASTG Sprint Backlog in Google Spreadsheets

2.4 Customer Requirements

2.4.1 Evolutionary Requirements

Lifecycle

Evolutionary requirements are requirements that change and evolve, as opposed to being specified up-front before the work begins. In iterative development, a basic set of requirements is discovered during the first three iterations or so. Most of the major requirements will generally be found by this time. Refinement of those requirements occurs after these iterations.

Focus

Early focus on requirements discovery should be on architecturally significant and risky requirements. During design the architect will need most of the non-functional or quality requirements (e.g. load, usability) and less of the functional requirements.

2.4.2 Capture Methods

Capturing requirements involves not only discussions with the customer, but demonstrating what we think the customer might want. The use of evolutionary requirements reduces the risk that a requirement is missing or wrong. Initially, much of the iteration time will be devoted to discovering and revising requirements.

Requirements should also consider the non-functional “ility” requirements: scalability, interoperability, modifiability, portability, validity, performance, reliability and security. These will become important during the development of the system architecture.

Requirements should always be visible and verified by the customer before implementation.

Other Work Products

Unified Modeling Language (UML) use-case diagrams can be nice for visualizing some of the major requirements, however, written use cases and details are a must. Use Case Analysis involves describing the high-level features of the software. The features can be broken down into sub-features, if necessary. Because each use case can be performed differently, use case scenarios or instances of a use case, can be considered to explain how each use case plays out.

For example: a *Purchase Item* use case can have a scenario involving new users and one involving existing accounts that already have a credit card on file.

In addition to the Requirements document, it may be advantageous to create early Top Ten Requirements (high-level only) and an analysis of the architecturally influential factors.

3 Software Development Guide

3.1 Software Development Procedure

This procedure is intended for use in all SIVO software projects. It should be tailored for the criticality of the implementation, such as doing more design when the changes are significant and may impact important pieces of the code. When specific information applies only to the GMI project, it is placed inside a gray box, as seen just below.

GMI

At the request of scientists, SIVO ASTG provides the software support to add new diagnostics and new components to the GMI code. Some of the diagnostics successfully implemented are: 3D tendencies, 3D emission, selection of a range of vertical levels for model outputs, selection of few species for model outputs, etc. The new modules introduced in the code are: Lightning, Micro-physical, Aerosol Dust Calculations.

ASTG follows rigorous guidelines to validate all the code modifications (with the assistance of scientists).

GMI: On-line References

The GMI Project has a website with an overview and guide materials, including the *Global Modeling Initiative: Tutorial and User's Guide*. A software developer should be familiar with this information. Refer to the Appendix for more information.

3.1.1 Understand the Requirements

Making the wrong changes to code is expensive. Understand the changes the requestor is asking for. If something is ambiguous, clarify it. Explain the changes to the requestor and make sure the requestor agrees with that understanding. Determine the level of documentation required for this development.

NOTE: *Do not accept requests from outside proper request channels. All requests must be approved and prioritized before work can begin.*

Refer to the In practice, ASTG employs the sprint backlog and product backlog on most projects and tracks these using Google spreadsheets, which are shared with all members of the project. The sprint backlog retains the list of completed items and shows the iteration cycle when it was completed (see example in Fig. 2.1 below)

Fig 2.1 ASTG Sprint Backlog in Google Spreadsheets

Customer Requirements *section for more information.*

3.1.2 Track the Request

All projects are tracked by the ASTG Project Manager in Google spreadsheets, using a separate spreadsheet for each project. These are normally updated by the project lead to show the task backlog and active tasks. All work requests must be approved by the PM, which will assign the

appropriate staff for each week. As individual subtasks are completed, they are ~~crossed out~~ and moved into the completed area of the spreadsheet.

3.1.3 Design the Software

Use appropriate methodologies and tools to create a design that meets the needs of the user and does not add burdens to future maintenance.

Refer to the UML and the Design Patterns sections for more information.

3.1.4 Check Out the Code

First, do preparations before making any changes. Tag the current repository. This enables an easier rollback to the starting point if you must back out of your changes.

GMI: Modifying Code

Become familiar with Chapter 6 of the *Global Modeling Initiative: Tutorial and User's Guide II* called "Making Changes," prepared by the SIVO staff. It is available on Modeling Guru at: <https://modelingguru.nasa.gov/docs/DOC-1417>.

Next, get a current copy of the source code from the Progress server. Do a **checkout** with Subversion or CVS, as appropriate, to get a fresh working copy of the files you need. If others are working on the same set of files, periodically use the **update** command to merge their revisions into your working copy.

*Refer to the **CVS** section at the end of this chapter for more information. Also see Chapter 8 of the GMI Tutorial and User's Guide II, referenced above, for practical examples and explanations of various CVS commands.*

3.1.5 Code the Changes

While making changes, be generous and clear with comments, updating previous comments as required. Apply the SIVO coding standards to newly developed modules, as appropriate.

For more information, refer to the [SIVO/ASTG Coding Standard](#) (on Modeling Guru).

GMI: Delimiting Code

If portions of code are limited to a particular chemical mechanism, use the **chem_mecha** variable to delimit these portions. Please note that you do not set this variable, it is already set for you. You just access and use it. To access it, just include "gmi_chemcase.h" at the appropriate place.

Refer to Chapter 6 of the *GMI: Tutorial and User's Guide* for more details.

3.1.6 Test and Validate the Code

Tests should be run as directed by the requestor. Unit and system tests developed during coding or already existing should be executed.

Before tests are deemed successful, requestor must be satisfied with results. Depending on the situation, this may require any or all of the following.

- Get new data from the scientists to run additional test cases.
- Run a plain diff on test cases to confirm consistent results.
- Run the tests out to a sufficiently long time period to test for bugs, such as memory leaks.
- When model physics change, so will test results—check with scientists to ensure all results remain within a reasonable range.
- Run the code under all configurations required (e.g. 12-16 currently in use for GMI). New datasets may be required for each configuration

When the code cannot be executed and tested separately, system level testing must be employed. Perform system level tests in each important configuration – multi-processor, multiple resolutions, etc. When near completion, add longer test runs to validate the stability and accuracy of the new/revised module.

To minimize errors that waste productivity, all system testing should include complete written descriptions of the experiment, including:

- Platform for execution
- Number of processors
- File naming conventions
- Environment Settings
- Set of test cases and verification criteria

To ensure repeatability of tests, a copy of the experiment description should be stored with the archived test data.

When possible, short initial test runs should be used to verify experiment setup. This avoids wasting computing resources on bad runs. For each experiment, confirm that the run completed normally and that all expected output files are produced.

Next, a first-cut analysis should be performed to ensure that:

- All necessary variables are written to file
- Record sizes are correct
- Values are reasonable through visual inspection (possibly using tools such as `ncdump`, `IDL`, or `vc.dat`)

Finally, perform an exhaustive verification of the output data, including any additional verification tests requested by the customer.

Refer to the Code Reviews and Test-Driven Development sections for more information.

GMI: Compiling

Since GMI employs four separate chemical mechanisms, ensure all four versions of the code will compile before proceeding to test the changes in the relevant version.

Refer to Chapter 6 of the *GMI: Tutorial and User's Guide* for more details.

GMI: File Names

Use the following convention in naming files:

`<model><out_typ>_<field>_<exp>_<time>_<attempt>.<dat>.<ftyp>`

where:

`<model>` is the model name and is one of

gms = GMI stratospheric chemistry model

gmt = GMI tropospheric chemistry model

gma = GMI aerosol model

gmc = GMI combined strat-trop chemistry model

`<out_typ>` is whether output is averaged or instantaneous

i = instantaneous values of the fields

a = averaged values of the fields

`<field>` is the met field used and is one of dao, ccm, gis, fvg, fvd

`<exp>` is the experiment name. Check with the requestor for an appropriate name.

`<time>` is the simulation period in months

`<attempt>` is how many times the same run have been attempted

`<dat>` is the variable written to file.

Currently we use: const, qj, flux, rst, rad, tend, etc.

`<ftyp>` is the file type i.e.

nc = netcdf

asc = ascii

Based on the above, a typical example would be:

`gmta_dao_fc-sloss100_Y0502-0503_2.const.nc`

where

`<model>` = gmt (GMI tropospheric chemistry model)

`<out_typ>` = a (averaged values of the files)

`<field>` = dao (DAO met fields)

`<exp>` = fc-sloss100 (testing strat lifetime of 100 days)

`<time>` = Y0502-0503 (Modeling Feb 2005 - Mar 2005)

`<attempt>` = 2 (Second attempt)

`<dat>` = const (Constituents output)

`<ftyp>` = nc (NetCDF format)

For the above example, the namelist file will be:

`gmta_dao_fc-sloss100_Y0502-0503_2.in`

3.1.7 Check In the Code

First, check the repository for changes. Before committing any changes to the repository, do a final update to merge changes and resolve any conflicts. Recompile and verify code still runs as expected. Next, tag the changes in source control and commit them. Decide on a descriptive **tag_name** to label this modification.

Proceed to commit the modified files to the repository. The CVS log should contain a detailed description of the modification including:

- why the modification was necessary
- who requested the modification
- any new variables added and what they represent
- any old variables removed

Immediately tag the code at the top-level using the same **tag_name** as above.

Refer to the CVS section for more information.

GMI: README/Tag File

Create a file in the \$GEMHOME/doc directory called README.<tag_name> and include in this file a clear description of the rationale for the tag. The description should be clear enough for somebody to be able to determine the purpose/use of this tagged version of the code.

3.1.8 Release the Code

During code development it may be necessary to maintain multiple release versions of software or create a separate version of the software for unstable development purposes. Maintaining revisions of software and documentation is an important part of project management. Since our department is using Subversion, CVS, Mercurial, and GIT, developers should become familiar with each of these code repository tools.

Before final delivery of a major software release, the following tasks must be completed:

- Complete a maintenance manual
- Update any current documentation to ensure consistency with new implementation
- Develop a tutorial document to provide step by step instructions to new users
- Post documentation to the website, as required
- Commit updated documentation to the appropriate repository

GMI: Update Versions Table

All new GMI releases must be added to the [GMI Version Numbers](#) document found on Modeling Guru. The new entry must show Version number, brief description, and the CVS tag name.

3.1.9 Doing Transition and Maintenance

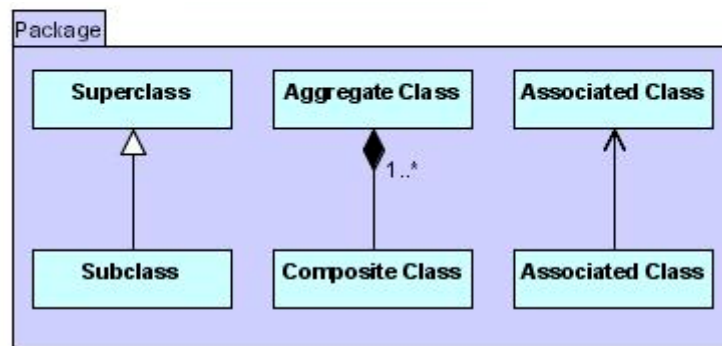
After the actual delivery of the release, additional support will normally be required to assist users implementing the new code and/or to address and problems that may arise. The following should be supplied to ease the transition process and educate users:

- Proactively communicate with key users periodically (weekly) for the first month to get feedback and answer questions
- Provide code walkthrough for NASA scientists/developers who will use the code
- Provide live demonstration to show new build process and new core configuration
- Be available for help desk assistance, as needed

3.2 Recommended Development Practices

3.2.1 UML

The following basic UML constructs are useful in creating **class** and **sequence diagrams**. UML has become the established way to diagram object-oriented code constructs and developers should know these basic constructs.



3.2.2 Design Patterns

Design Patterns are useful in constructing code because they solve problems that are already well known in software. Most developers should be familiar with the most commonly used patterns. A few of these include Adapter, Façade, Abstract Factory, Decorator, Factory Method, Strategy, Template Method, Singleton, Observer, and Composition.

3.2.3 Inspections

An inspection provides the opportunity to disseminate information, give feedback, find errors, and make corrections earlier in a project. It allows one to assess and improve the project's quality and progress and once learned, it complements other QA techniques.

An inspection can be conducted at any stage from requirements to code. Each reviewer will spend some time familiarizing himself with the author's work (a checklist can be used). Afterwards an inspection meeting is held, generally lasting no longer than two hours at a time. The

author paraphrases the work and explains the logic. Errors and solutions (if discussed) should be recorded.

The item being examined in a review should still be considered under development. Designs should therefore be reviewed before the design is fully implemented and code should be reviewed before the product is released. They should focus on error detection rather than error correction and pay attention to problem areas in the past.

3.2.4 Code Reviews

A code review is a special inspection to examine source code. Two or more reviewers read the author's code independently before meeting. Qualitative aspects such as design, style, readability, maintainability and efficiency can be commented on. Positive aspects should be commended and encouraged.

When the reviewers are finished, a meeting can be held which focuses specifically on the problems discovered by the code readers and lasts one or two hours. The author should fix the problems identified by the reviewers.

ASTG's Quality Engineer regularly performs random Coding Standards Reviews with a standardized checklist to ensure all programmers are using the same conventions. These may be done in conjunction with a complete code review, but are often conducted separately.

3.2.5 Pair Programming

Pair programming is like a real-time code review where two programmers take turns writing code together. It has been shown to reduce the number of defects in a code and to increase knowledge sharing among developers. Partners can be exchanged after an iteration. It is recommended that two inexperienced developers not be paired together.

3.2.6 Automated Documentation

Documentation systems can generate reference manuals from a set of documented source files, avoiding duplication of effort and making it much easier to keep the documentation consistent with the source code. The open source **Doxygen** system can be used on C++, C, Java, Fortran, IDL, and Python, while **ProTeX** (developed at GSFC) is also available for Fortran code. Both will produce LaTeX documents from specially commented source code. Doxygen can output several other common document formats as well.

3.2.7 Test-Driven Development

Test-driven development (TDD) is the practice of creating unit tests before writing the code (e.g. test-first development). The units tests can all be run automatically and return only a pass or fail indication. Unlike traditional programming, the test code is being written **before** the actual code. This requires some practice before getting comfortable.

TDD and test-first development is recommended for use because it provides several benefits. Testable code obviously makes for safer development and changes. Also tests are more likely to be written if they are done first because it is less burdensome and more sustainable than writing them after the code is working. Finally, creating the tests requires thinking through how to use the public interface, improving the actual development.

A commonly used testing framework tool called XUnit provides a wide range of test functionality. The following is a short list of some relevant ones:

- **JUnit** for Java
- **CppUnit** for C++
- **pFUnit** for Fortran (developed at SIVO, and published on [SourceForge](#))

The process of using TDD works like this:

1. Write a test and invent the interface.
2. Run the test to make sure it fails.
3. Make small changes to the development code to get it to run and compile.
4. Run the test to make sure it passes.
5. Refactor the code to remove duplication.

TDD is becoming the norm for new development within SIVO, and is the subject of many posts and documents on Modeling Guru.

3.2.8 CVS Tips and Practices

ASTG team members often work on portions of the code at the same time and introduce changes that may be in conflict with each other. To avoid any potential problem, the team follows some well-defined procedures and use CVS to manage and keep track of all the changes made in the code. ASTG also uses other revision control systems, such as Subversion and Mercurial, but the basic concepts are the same.

CVS can be used very effectively for change control management; however, there are a few standard practices that make life easier for both you and others on your team. Apply the following principles.

Tag Early, Tag Often

- Make a tag whenever you do anything interesting (especially before and after merging code). Tags are useful for reproducing a particular configuration of files. It never hurts to create a tag and it could save a lot of work later if something goes wrong.
- Choose names that will not conflict since tags are global. Adhere to whatever standards a project may already have.

Modifying Code

- When making modifications, change as few lines as possible (e.g. do not re-indent the whole file just for fun). This should be done on a separate change.

Unstable Development and Branching

- Use branches for doing lengthy, unstable development. You can back up your code without breaking other peoples' code. Module pages should document branches they are using.

- When merging code back to the main branch, merge all changes to the main trunk, tag it and commit it.
- The CVS trunk should generally be used for ongoing development and the code should compile and be reasonably stable and at least minimally tested.

Stable Development and Branching

- Branches can be created for stabilizing the entire code base in preparation for an official release. Commits to these branches should only fix important bugs and not add new functionality. (Don't forget to apply the bug fixes to the main trunk)

4 Production Run Guide

GMI scientists request production runs to be made for various reasons including checking the effect of a particular parameter or a number of parameters on certain chemical species concentrations. The data from these runs may be used for producing manuscripts, for producing data for presentations at meetings, and for submission to global community experiments evaluating chemical budgets of similar models.

The runs requested usually will require one of four chemical mechanisms: troposphere, aerosol, stratosphere, and combined stratosphere-troposphere (“combo”). Tracer runs and radionuclide runs are sometimes requested as well. The different meteorological fields that are used for the runs at present are GEOS5-DAS, GEOS4-DAS, FVGCM, and GISS fields.

4.1 Production Run Procedure

GMI scientists will request that ASTG GMI team members perform production runs or sensitivity experiments. All run requests from the scientists will be directed to the ASTG GMI project leader who will then pass the proper information to ASTG personnel.

4.1.1 Evaluate the Request

Consider the following questions before starting the task:

- What chemical mechanism is requested?
- When does the requester need the results?
- Is the requested deadline reasonable?
- Do I need to make changes to the current version of the code to accomplish this?
- If I need to make changes, are they substantial?
- Do I have other work to complete first?

4.1.2 Get the Requirements

Determine the correct customer requirements, model configuration, and input datasets to be used. If any questions arise during this process, the ASTG staff member working on the request will contact the ASTG GMI project leader. Then, depending on the question and the run request, the ASTG GMI project leader or the ASTG staff member will contact the customer for more information.

4.1.3 Prepare for the Run

When generating the namelist file and run scripts for a particular run, please follow the established GMI file and directory naming convention, as outlined in the GMI User’s Guide. The user’s guide also contains step-by-step instructions about how to run the code and a list and description of all namelist variables. Input files used for any run are to be placed in the appropriate input directory under the gmidata2/ structure on Discover and Dirac. When adding files to the input/ directory tree please add a description of the file to the README file in the appropriate subdirectory.

GMI: Directory Naming

Create a unique directory for the run on dirac (or appropriate mass storage system) in
`/pub/gmidata/<physics>/<sub-physics>`

where

`<physics>` = one of `gmia`, `gmic`, `gmis`, `gmit`

`<sub-physics>` = one of `chem`, `rnuc`, `trac` for `gmit`, OR blank for `gmia`, `gmic`, `gmis`

In naming the directory, use the convention described in paragraph 3.1.6 in the above section, i.e. the directory should be named:

`<model><out_typ>_<field>_<exp>_<time>_<attempt>`

Add necessary modifications to the namelist file and recheck. Check for the problem name, total number of days, and number of processors needed. These are the major variables that need to be changed from a test run to a production run.

Output data from the runs should be temporarily placed in a directory in the ASTG staff member's personal space, whether on the local machine or on Dirac. No output is to be placed in the `gmidata2/output/` directory on the mass storage system/anonymous ftp site (Dirac) until the GMI scientists have given their approval.

4.1.4 Examine and document the outputs

The simulation outputs first are to be examined by ASTG GMI staff to make sure there are no obvious errors or problems. Then the requestor is given the location of the output data for review and analysis, so they can decide if the data are correct. Once the data are approved, the files can be moved to the appropriate permanent directory under `gmidata2/output/` on Dirac. Put the namelist input file, batch script, and all the output files on the mass storage system in the conventional structure documented in the GMI User's Guide. If necessary, commit the version of the code with a tag name using CVS. If the customer needs further assistance, provide help as necessary.

After completing a full production run and following approval by the GMI scientists, properly document the experiment to help construct a useful resource for others. Write a README file if needed or modify an existing README file in the directory on Dirac in which the data are kept. In the README file, include the following information about the runs if available:

- Date of the data was placed in the output/ directory
- Name or initials of the person who performed the run
- Description of the unique features of this run
 - Scientific reason for running it
 - Changes made to the code or namelist file
- CVS tags used

You also need to update the run documentation in the [GMI Production Run Spreadsheet](#) on Modeling Guru. This spreadsheet contains two sheets, one contains the full details and the other is a condensed version for quick reference. Use the existing entries as examples to update the spreadsheet.

Note that larger tasks—those that require extensive code changes, or testing that may take weeks or months to complete—can be divided into subtasks. When a larger run is requested, follow these suggestions:

- Divide the complete work into small achievable goals
- At the beginning of the day, use half an hour to evaluate what you planned to do the day before and whether you completed it or not. If completed, set the next goal. If not completed, then evaluate the reasons why it was not completed. Take steps to make that pending job complete. If you need help, contact team members and discuss the problems with them.
- At the end of each day, evaluate the day's work before you leave the office and make notes of what was achieved and what was not achieved on that particular day.
- Note the deliverable date. If you determine that the runs will not be ready for the requested deliverable date, list and evaluate the reasons why and speak with the ASTG GMI project leader as soon as possible, who will contact the requestor if necessary.

Follow the above guidelines for delivering the smaller tasks and verifying the effectiveness of those smaller tasks.

5 Level II User Support Guide

5.1 Objective

Level II Support provides users of the NCCS and NAS HEC supercomputers with technical support assistance that is beyond the expertise of the Level I Help Desk. ASTG provides support for applications programming problems (i.e., GMI configuration), and provides guidance in using high-performance tools, such as debuggers, profilers, programming languages, queuing systems, environment modification, and other issues that require advanced programming and troubleshooting expertise. Virtually every SIVO ASTG Analyst may be asked to provide Support Assistance in their area of expertise.

A brief support checklist is at: [Level II Support - Basic Guidelines](#) on NASA Modeling Guru.

5.2 Technical Approach

5.2.1 Track the Request

Requests for assistance generally originate from the NCCS or NAS Help Desk and are tracked using the NCCS ticketing system. They may also be forwarded directly from SIVO management based on requests they receive from users. ASTG staff have the option of creating tickets themselves for such requests.

If a referred request arrives without adequate information, such as user contact information, location of user's application, platform in use, etc., analysts will determine if the request should be handed back to Level I to ask them to obtain more details, or will contact the user themselves.

The NCCS ticketing system uses the Numera Footprints application. At a minimum, the following items must be entered into the Footprints database:

- Requester's contact information
- Date of request
- Description of the feature or problem
- Developer assigned to resolve the issue
- Any other relevant information

In addition, the ticket information (number, title, date received) must be recorded in "Level-II-Support<time-period>" Google spreadsheet. Do not forget to evaluate the magnitude of the ticket.

Tickets will remain open until the requester is satisfied with the resolution and gives his/her permission to close the ticket. The system will periodically remind each developer if tickets are still open. Each developer is responsible for the tickets assigned to them, and may only close their own tickets.

5.2.2 Diagnose the Problem

The user should be contacted within three hours after receiving a ticket, both as a courtesy and to verify that you understand the requestor's problem. Given adequate information about the user's

problem, the analyst will evaluate the cause, and determine if it relates to a question that can be immediately answered.

For more advanced problems that relate to errors in code, the analyst will determine if they should try to run their own copy of the user's code, or if they should attempt to guide the user in experiments to try on their own.

Special arrangements may need to be made if there is a need to obtain the user's code and they do not wish to make it world readable. Such cases may require further interaction with NCCS or NAS Systems Administrators.

5.2.3 Determine the Resolution

If the question can be answered immediately, users will be given a solution, usually via email, but sometimes by phone or even via one-to-one training sessions if needed. If possible, the analyst may cite references in NCCS, NAS, SIVO, or Modeling Guru web pages or other educational websites that may offer the user more details they can study.

If the analyst determines the problem relates to a system malfunction that must be addressed by NASA Systems Staff or by the vendor, they will inform the user, and refer/reassign the ticket to the appropriate staff.

If a feasible solution is offered or suggested, the analyst will ask the user to report the results. This may take some time, as many applications may run for several hours, and may wait in the queues for several hours depending upon the current system loads. The analyst will attempt to ensure an acceptable solution was found, and in some cases may need to contact the user to follow up. If the solution could be useful to others, it should be summarized and posted to the appropriate community on Modeling Guru.

If a problem is extremely complex, and requires special skills of a specific analyst, or is thought to become a long-term project, the analyst will refer the issue to Contractor and/or SIVO management for further evaluation as to how to dedicate resources to its resolution.

Always keep the user informed on the progress of your work. Record all communications (email, phone, face-to-face meetings) with the user on the ticketing system.

5.2.4 Closing a Ticket

As soon as an issue is resolved, close the ticket or set the ticket to auto-close. If you referred the problem to NASA Systems Staff or a vendor, reassign the ticket to the appropriate staff. Always inform the user of any action you take on a ticket. Remember to update the Google Level-II spreadsheet (status, closing iteration period) and if necessary, write down any pertinent information.

If an analyst recognizes a trend in repeated requests for the same type of assistance or sees a lack of web materials that may be of use in educating the user community, they may contact Contractor and/or SIVO management to recommend the preparation of more knowledge base materials to host on Modeling Guru, and possibly user training classes.

6 Web Site Development Guide

6.1 Objective

SIVO maintains a website that caters to internal and external customers. It provides the public with an overview of the key projects carried out within SIVO, and also provides information to customers about important modifications to models maintained by SIVO.

NASA has many regulations concerning website development, so all changes must be approved and conform to NASA standards before being published to the server.

6.2 Development Procedure

6.2.1 Preparations

Before creating a site, go to <http://webmaster.gsfc.nasa.gov> and familiarize yourself with the NASA Web Policies. Sites must meet 508 requirements, Children's Online Privacy Protection Act, NASA Privacy policy, NASA Internet Publishing Guidelines, NASA Logo Policy, NASA Banner policy, NASA Scientific and Technical Information guidelines, Sensitive But Unclassified information policies, Export Control policies, GSFC web requirements, log retention policy and others.

Discuss with SIVO management and personnel the requirements for the web site. Create a storyboard before constructing site. Once structure is agreed upon, move on to creating actual site.

6.2.2 Creating Pages

1. CSS templates are available for the NASA Portal design. The templates and instructions are available at <http://portalcss.gsfc.nasa.gov/>
2. Follow the instructions for how to generate a NASA Portal set of web pages and images. You will be required to select a set of templates (Basic HTML Package, Server Side Includes Package, or PHP Package).
3. Once you have decided on the template package, download it to your local system.
4. Using a web development software package (e.g. *Dreamweaver*) or code language, begin developing the individual pages and populate the web site with content provided by SIVO staff and management. Follow the instructions for modifying the Navigation Bar and creating new pages and inserting content. Follow the rules of XHTML for clean code and most CSS mistakes will be avoided.
5. Develop a banner image (in *Adobe Photoshop*, *Fireworks*, etc) to replace the placeholder banner image.
6. Test the site on a test server. A new test server is available at <http://panini-dev.gsfc.nasa.gov>. Contact the webserv system administrator for access.

6.2.3 Web Site Approval Process

1. For web sites on Panini, virtual hosting is required. To receive a domain name, send an email to the CSO (presently, George Rumney) requesting a domain registration name. Once this is received, contact NCCS User Services to request the creation of a directory on Panini. You must let them know if you are using scripting so that the server can be configured to accept it. A ticket will be generated and assigned to the system administrator for Panini (presently, Jasaun Neff). Once he has completed the action, the ticket will be closed out and the user will be notified of the directory name, user name, password, and other account information.
2. Ensure the site is compliant with NASA web policy and Federal law by registering the web site in the Agency Web Registry System (AWRS) at <https://webregistration.larc.nasa.gov>. Please note that an account is required to access this system. An application for access is also available at the address above.
3. Begin the web port waiver process, using the reference found at:
<http://webmaster.gsfc.nasa.gov/policy/gsfcp/waiver/index.html>
Review the web port waiver process flowchart for step-by-step instructions
<http://webmaster.gsfc.nasa.gov/policy/gsfcp/waiver/flowchart.html>
4. Determine whether or not the site contains Scientific and Technical Information (STI) and/or Export Controlled information. If so, fill out the GSFC STI Public Disclosure Export Control Checklist (GSFC 25-49) and the NASA Scientific and Technical Document Availability Authorization (DAA) form (NASA 1676).

The NASA form is available at <http://server-mpo.arc.nasa.gov/services/NEFS> and the GSFC form is at <http://gdms.gsfc.nasa.gov/home.jsp>. To find the GSFC form, look under LIBRARIES ⇒ Forms Master List ⇒ Goddard Forms GSFC. See the SIVO Admin office (presently, Ruby Adams) for further assistance, and to get the proper GSFC routing cover sheet. Allow 2 weeks for processing.
5. Fill out web waiver form found at:
<http://webmaster.gsfc.nasa.gov/policy/gsfcp/waiver/flowchart.html>
Allow 1 week for processing.

6.2.4 Publishing

Once the web port waiver has been granted, post the site. This web port waiver must be renewed yearly through AWRS. The actual location of the files is on webserv at var/www/sivo.

An easy way to post files to webserv with ssh is by using a program called *Fugu*. After entering user name and password, you can upload files with a drag and drop from the desktop directory structure to webserv. This can also be done through the terminal using Unix commands. You must use the LDAP dual-authentication user id.

6.3 GMI Web Site Special Operations

6.3.1 Schedule

GMI web site usage statistics are run on the first day of each month (or, if the first day of the month falls on a weekend or holiday, the next business day).

6.3.2 Procedures

To **UPDATE** the GMI webalizer statistics:

1. Log into webserv
2. `cd to /var/www/gmi/webalizer`
3. Execute the command: `sh run-webalizer`

To **VIEW** the GMI webalizer statistics:

1. Open a web browser and go to <http://gmi.gsfc.nasa.gov/webalizer/>
2. Click on the links for various months to view the detailed statistics. Note the time and date at the top of the page indicating the last time statistics were updated.

To **SAVE** a GMI webalizer report:

1. Click on the link to the latest month in the table.
2. Save this file as a PDF using the following naming convention: `usage_MM_YYYY`

To **SUBMIT** a GMI webalizer report:

Send the PDF file as an attachment to the GMI Project Manager (currently Susan Strahan at strahan@prometheus.gsfc.nasa.gov)

To **DELETE** a GMI web site log:

1. Log into webserv.
2. `cd to /var/www/gmi/logs/`
3. Execute the command: `rm access.log`

6.4 Print-Protecting GMI PDFs

Susan Strahan has requested that all PDFs posted on the GMI web site be password restricted for printing or editing. Adobe Acrobat (full version) has the ability to do this.

To password restrict a PDF file:

1. Open the file in Adobe Acrobat.
2. Choose Document > Security > Restrict Opening and Editing.
3. Check the box that reads "Use a password to restrict printing and editing of the document and its security settings."
4. Enter the password in the Permissions Password field.

5. Choose “None” in the drop down boxes for “Printing allowed” and “Changes allowed.”
6. Check the box that reads “Enable text access for screen reader devices for the visually impaired.”
7. Confirm the password when prompted.
8. Save the document.

7 Appendix

7.1 References

7.1.1 GMI Project

- GMI Documentation
http://gmi.gsfc.nasa.gov/home_models.html
- *Global Modeling Initiative: Tutorial and User's Guide*
<https://modelingguru.nasa.gov/clearspace/docs/DOC-1417>.
- Aerosol Dust Calculations
http://gmi.gsfc.nasa.gov/models/AerDust_Computations.pdf

7.1.2 CVS

- CVS Home
<http://ximbiot.com/cvs/manual/index.html>
- CVS
http://www.delorie.com/gnu/docs/cvs/cvs_toc.html
- CVS
http://web.mit.edu/afs/athena.mit.edu/project/gnu/doc/html/cvs_toc.html
- Versioning and Branching With CVS
<http://www2.cs.utah.edu/impulse/cvs/index.html>
- The CVS FAQ
<http://www.ece.cmu.edu/~ece749/cvsfaq.htm>
- CVS Introduction Lecture
<http://www.cdt.luth.se/~peppar/presentations/cvs/>
- Maintaining the ChangeLog
http://www.gnu.org/software/guile/changelogs/guile-changelogs_toc.html

7.1.3 Website Development

- SIVO Website
<http://sivo.gsfc.nasa.gov>
- NASA Web Policies
<http://webmaster.gsfc.nasa.gov>
- NASA Templates and Graphics
<http://portalgraphics.gsfc.nasa.gov/>
<http://portalcss.gsfc.nasa.gov/templates/index.php>
- NASA Web Registry
<https://webregistration.larc.nasa.gov>

7.1.4 Software Development

- Agile Alliance Home
<http://agilealliance.org/intro>

7.2 Common Tools

Whenever necessary, ASTG team members may design and implement utility tools to facilitate the work of scientists (pre- and post-processing analyses) and to simplify the model implementation. An example of tools is the netCDF utility library that has been used in the GMI code and in various applications to manipulate netCDF files.

Recommended tools for ASTG software development include the following:

Name	Category	Description
Eclipse	Development Environment	Open source, cross-platform integrated development environment popular for Java, C++ and also Fortran. The current version is has a learning curve but it offers powerful integrated features such as a visual editor, CVS, debugging, UML design, and testing. http://www.eclipse.org
Photran	Development Environment	Open source plug-in for Eclipse that provides a Fortran development environment. http://www.eclipse.org/photran
vi / emacs	Development Environment	For basic text editing, vi and emacs are the standard tools. Today vi ("Visual Implementation" of an older Unix editor) may be considered the default, since it is the only editor that will be installed on every UNIX system. Emacs stands for Editor MACroS. It has become a de facto standard alongside vi, but is more than just a text editor. It can be used as a complete system for development, communications, and file management.
CppUnit	Testing	CppUnit is the C++ port of the popular JUnit framework for Java unit testing. Test output is in XML or text format for automatic testing, and GUI based for supervised tests. Tests in CppUnit can be run automatically. They are easy to set up and once you have written them, they are always there to help you keep confidence in the quality of your code. http://cppunit.sourceforge.net
pFUnit	Testing	pFUnit is a unit testing framework for Fortran, developed in part by the ASTG.

		http://sourceforge.net/projects/pfunit
CVS	Configuration Management	<p>The Concurrent Versions System (CVS) is not a complete configuration management system. It is a source control system that allows tagging of specific versions in its repository. This lets users recall any previously archived version of the code. It prevents developer conflicts by making each check-out their own version, and helps merge changes when they're done.</p> <p>http://ximbiot.com/cvs/wiki/index.php</p>
Poseidon	Design	<p>Commercial product built on an open source engine that provides the ability to create numerous types of UML diagrams, as well as forward and reverse-engineer code. The community edition is free.</p> <p>http://www.gentleware.com/products.html</p>
Perl scripts	Automation	<p>Perl is a high-level programming language that is fairly easy to learn. Perl's process, file, and text manipulation facilities make it particularly well-suited for tasks involving quick prototyping, system utilities, software tools, system management tasks, and web programming.</p>
Doxygen	Documentation	<p>Open source documentation system for C++, C, Java, Python, IDL, Fortran, and to some extent PHP and C#. It can generate an on-line documentation browser (in HTML) and/or an off-line reference manual (in common document formats) from a set of documented source files.</p> <p>http://www.doxygen.org</p>
ProTeX	Documentation	<p>Proprietary GSFC perl script that will generate LaTeX documentation from specially commented source code.</p> <p>http://gmao.gsfc.nasa.gov/software/protex/</p>

7.3 NASA STI Approval for Journal/Conference Papers

Before submitting a scientific article to a conference or journal, the author must complete the GSFC STI Public Disclosure Export Control Checklist (GSFC 25-49) and the NASA Scientific and Technical Document Availability Authorization (DAA) form (NASA 1676).

The NASA form is available at <http://server-mpo.arc.nasa.gov/services/NEFS> and the GSFC form is at <http://gdms.gsfc.nasa.gov/home.jsp>. To find the GSFC form, look under LIBRARIES ⇒ Forms Master List ⇒ Goddard Forms GSFC. See the SIVO Admin office (presently, Ruby Adams) for further assistance, and to get the proper GSFC routing cover sheet. Allow 2 weeks for processing.